10/25/2021

DSA LAB Session 2

LAB Tasks

Fassih ur rehman

REG NO : 200901054

Collections documentation

My work

1. Deque

list-like container with fast appends and pops on either end
Deques are a generalization of stacks and queues (the name is pronounced “deck” and is short for “double-ended queue”). Deques support thread-safe, memory efficient appends and pops from either side of the deque with approximately the same O(1)

2. Counter

Dictionary ubclass for counting hashable objects
A `Counter` is a **`dict`** subclass for counting hashable objects. It is a collection where elements are stored as dictionary keys and their counts are stored as dictionary values. Counts are allowed to be any integer value including zero or negative counts

Deque objects support following:

3. POP()

Remove and return an element from the right side of the deque. If no elements are present, raises an IndexError.

4. append(x)

Add *x* to the right side of the deque.

5. appendleft(*x*)

Add *x* to the left side of the deque.

6. clear()

Remove all elements from the deque leaving it with length 0.

7. copy()

Create a shallow copy of the deque.

8. Extend

Extend the right side of the deque by appending elements from the iterable argument.

9. Extend

Extend the right side of the deque by appending elements from the iterable argument.

10. Extendleft

Extend the left side of the deque by appending elements from *iterable*. Note, the series of left appends results in reversing the order of elements in the iterable argument.

11. Insert(I, x)

Insert *x* into the deque at position *i*.

12. **remove**(*value*)
    Remove the first occurrence of *value*. If not found, raises a `ValueError`.
13. **reverse**()
    Reverse the elements of the deque in-place and then return `None`.

14. **rotate**(*n=1*)
    Rotate the deque *n* steps to the right. If *n* is negative, rotate to the left.

15. **maxlen**
    Maximum size of a deque or `None` if unbounded.

COUNTER Objects

1 elements()

Return an iterator over elements repeating each as many times as its count

2. most common([n])

Return a list of the *n* most common elements and their counts from the most common to the least. If *n* is omitted or `None`, `most_common()` returns *all* elements in the counter.

3. total()
Compute the sum of the counts.